What Is Claimed Is:

1. A method for selectively unmarking load-marked cache lines during transactional execution, wherein load-marked cache lines are monitored during transactional execution to detect interfering accesses from other threads, the method comprising:

encountering a release instruction during transactional execution of a block of instructions within a program, wherein changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution completes without encountering an interfering data access from another thread; and

in response to the release instruction, modifying the state of cache lines, which are specially load-marked to indicate they can be released from monitoring, to account for the release instruction being encountered;

wherein modifying the specially load-marked cache lines can cause the specially load-marked cache lines to become unmarked.

2. The method of claim 1,

wherein a specially load-marked cache line contains a release value indicating how many release instructions need to be encountered before the cache line can become unmarked;

wherein modifying the specially load-marked cache line involves decrementing the release value; and

wherein if decrementing the release value causes the release value to become zero, the cache line becomes unmarked.

3. The method of claim 2, wherein decrementing release values for all specially load-marked cache lines involves incrementing a global counter, wherein release values are initialized with respect to the global counter.

4. The method of claim 1, wherein upon encountering a load instruction during the transactional execution, the method further comprises:

performing the corresponding load operation; and

if the load instruction is a monitored load instruction, load-marking a corresponding cache line to facilitate subsequent detection of an interfering data access to the cache line from another thread;

wherein if the load instruction additionally specifies that the corresponding cache line can be released from monitoring during transactional execution, load-marking the cache line involves specially load-marking the cache line to indicate that the cache line can be released from monitoring after either an explicit or implicit number of release instructions have been encountered.

5. The method of claim 1, wherein if an interfering data access from another thread is encountered during transactional execution, the method further comprises:

discarding changes made during the transactional execution; and

attempting to re-execute the block of instructions.

6. The method of claim 1, wherein if transactional execution completes without encountering an interfering data access from another thread, the method further comprises:

committing changes made during the transactional execution to the architectural state of a processor; and

resuming normal non-transactional execution of the program past the block of instructions.

7. The method of claim 1, wherein an interfering data access can include:

a store by another thread to a cache line that has been load-marked by a thread; and

a load or a store by another thread to a cache line that has been store-marked by the thread.

8. An apparatus that selectively unmarks load-marked cache lines during transactional execution, wherein load-marked cache lines are monitored during transactional execution to detect interfering accesses from other threads, the apparatus comprising:

an execution mechanism configured to execute a release instruction during transactional execution of a block of instructions within a program, wherein changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution completes without encountering an interfering data access from another thread; and

an updating mechanism, wherein in response to the release instruction, the updating mechanism is configured to modify the state of cache lines, which are specially load-marked to indicate they can be released from monitoring, to account for the release instruction being encountered;

wherein while modifying the specially load-marked cache lines, the updating mechanism can cause the specially load-marked cache lines to become unmarked.

9. The apparatus of claim 8,

wherein a specially load-marked cache line contains a release value indicating how many release instructions need to be encountered before the cache line can become unmarked;

wherein while modifying the specially load-marked cache line, the updating mechanism is configured to decrement the release value; and

wherein if decrementing the release value causes the release value to become zero, the cache line becomes unmarked.

10. The apparatus of claim 9, wherein decrementing release values for all specially load-marked cache lines involves incrementing a global counter, wherein release values are initialized with respect to the global counter.

11. The apparatus of claim 8, further comprising a load-marking mechanism, wherein upon encountering a load instruction during the transactional execution, the load-marking mechanism is configured to:

perform the corresponding load operation; and

if the load instruction is a monitored load instruction, to load-mark a corresponding cache line to facilitate subsequent detection of an interfering data access to the cache line from another thread;

wherein if the load instruction additionally specifies that the corresponding cache line can be released from monitoring during transactional execution, the load-marking mechanism is configured to specially load-mark the cache line to indicate that the cache line can be released from monitoring after either an explicit or implicit number of release instructions have been encountered.

12. The apparatus of claim 8, wherein if an interfering data access from another thread is encountered during transactional execution, the execution mechanism is configured to:

discard changes made during the transactional execution; and to

attempt to re-execute the block of instructions.

13. The apparatus of claim 8, wherein if transactional execution completes without encountering an interfering data access from another thread, the execution mechanism is configured to:

commit changes made during the transactional execution to the architectural state of the processor; and to

resume normal non-transactional execution of the program past the block of instructions.

14. The apparatus of claim 8, wherein an interfering data access can include:

a store by another thread to a cache line that has been load-marked by a thread; and

a load or a store by another thread to a cache line that has been store-marked by the thread.

15. A method for selectively unmarking load-marked cache lines during transactional execution, wherein load-marked cache lines are monitored during transactional execution to detect interfering accesses from other threads, the method comprising:

encountering a commit-and-start-new-transaction instruction during transactional execution of a block of instructions within a program, wherein

changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution completes without encountering an interfering data access from another thread; and

in response to the commit-and-start-new-transaction instruction, modifying load-marked cache lines to account for the commit-and-start-new-transaction instruction being encountered;

wherein modifying the load-marked cache lines causes normally load-marked cache lines to become unmarked, while other specially load-marked cache lines may remain load-marked past the commit-and-start-new-transaction instruction;

whereby a new transaction can be immediately started with some cache lines already load-marked, thereby allowing a cache line to be monitored across consecutive transactions.

16. The method of claim 15,

wherein a specially load-marked cache line contains a checkpoint value indicating how many checkpoint-and-commit instructions need to be encountered before the cache line can become unmarked;

wherein modifying the specially load-marked cache line involves decrementing the checkpoint value; and

wherein if decrementing the checkpoint value causes the checkpoint value to become zero, the cache line becomes unmarked.

17. The method of claim 16, wherein decrementing release values for all specially load-marked cache lines involves incrementing a global counter, wherein release values are initialized with respect to the global counter.

18. The method of claim 15, wherein upon encountering a load instruction during the transactional execution, the method further comprises:

performing the corresponding load operation; and

if the load instruction is a monitored load instruction, load-marking a corresponding cache line to facilitate subsequent detection of an interfering data access to the cache line from another thread;

wherein if the load instruction additionally specifies that multiple checkpoint-and-commit instructions need to be encountered before the cache line can become unmarked, and load-marking the cache line involves specially load-marking the cache line to indicate that multiple checkpoint-and-commit instructions need to be encountered before the cache line can become unmarked.

19. The method of claim 15, wherein if an interfering data access from another thread is encountered during transactional execution, the method further comprises:

determining if the transactional execution was initiated by a commit-and-start-new-transaction, and if so executing alternative code to complete the transactional execution in the presence of the interfering data access; and

otherwise, discarding changes made during the transactional execution, and attempting to re-execute the block of instructions.

20. The method of claim 15, wherein if transactional execution completes without encountering an interfering data access from another thread, the method further comprises:

committing changes made during the transactional execution to the architectural state of the processor; and

resuming normal non-transactional execution of the program past the block of instructions.

21. The method of claim 15, wherein an interfering data access can include:

a store by another thread to a cache line that has been load-marked by a thread; and

a load or a store by another thread to a cache line that has been store-marked by the thread.

22. An apparatus that selectively unmarks load-marked cache lines during transactional execution, wherein load-marked cache lines are monitored during transactional execution to detect interfering accesses from other threads, the apparatus comprising:

an execution mechanism configured to execute a commit-and-start-new-transaction instruction during transactional execution of a block of instructions within a program, wherein changes made during the transactional execution are not committed to the architectural state of a processor until the transactional execution completes without encountering an interfering data access from another thread; and

an updating mechanism, wherein in response to the commit-and-start-new-transaction instruction, the updating mechanism is configured to modify load-marked cache lines to account for the commit-and-start-new-transaction instruction being encountered;

wherein while modifying the load-marked cache lines, the updating mechanism can cause normally load-marked cache lines to become unmarked,

while other specially load-marked cache lines may remain load-marked past the commit-and-start-new-transaction instruction;

whereby a new transaction can be immediately started with some cache lines already load-marked, thereby allowing a cache line to be monitored across consecutive transactions.

23. The apparatus of claim 22,

wherein a specially load-marked cache line contains a checkpoint value indicating how many checkpoint-and-commit instructions need to be encountered before the cache line can become unmarked;

wherein while modifying the specially load-marked cache line, the updating mechanism is configured to decrement the checkpoint value; and

wherein if decrementing the checkpoint value causes the checkpoint value to become zero, the cache line becomes unmarked.

24. The apparatus of claim 23, wherein decrementing release values for all specially load-marked cache lines involves incrementing a global counter, wherein release values are initialized with respect to the global counter.

25. The apparatus of claim 22, further comprising a load-marking mechanism, wherein upon encountering a load instruction during the transactional execution, the load-marking mechanism is configured to:

perform the corresponding load operation; and

if the load instruction is a monitored load instruction, to load-mark a corresponding cache line to facilitate subsequent detection of an interfering data access to the cache line from another thread;

wherein if the load instruction additionally specifies that multiple checkpoint-and-commit instructions need to be encountered before the cache line can become unmarked, the load-marking mechanism is configured to load-mark the cache line to indicate that multiple checkpoint-and-commit instructions need to be encountered before the cache line can become unmarked.

26. The apparatus of claim 22, wherein if an interfering data access from another thread is encountered during transactional execution, the execution mechanism is configured to:

determine if the transactional execution was initiated by a commit-and-start-new-transaction, and if so to execute alternative code to complete the transactional execution in the presence of the interfering data access; and

otherwise, to discard changes made during the transactional execution, and to attempt to re-execute the block of instructions.

27. The apparatus of claim 22, wherein if transactional execution completes without encountering an interfering data access from another thread, the execution mechanism is configured to:

commit changes made during the transactional execution to the architectural state of the processor; and to

resume normal non-transactional execution of the program past the block of instructions.

28. The apparatus of claim 22, wherein an interfering data access can include:

a store by another thread to a cache line that has been load-marked by a thread; and

a load or a store by another thread to a cache line that has been store-marked by the thread.